

JS Level 0



Введение

Это занятие может вам показаться достаточно простым, в чём-то даже "очевидным".

Оно является вводным и содержит ряд ключевых идей, которые мы будем использовать далее, поэтому изучите приведённую в этом занятии информацию внимательно (это относится и к остальным занятиям).



Цели



Цели

Мы (команда Alif Skills) разработали линейку практикумов. В этих практикумах мы делимся с вами нашим опытом и предоставляем вам тренажёры, на которых вы сможете на практике отработать полученные навыки.



Цели

Цель практикумов JS – позволить вам стать FullStack-разработчиком.

Кто такой FullStack-разработчик? Это специалист, обладающий навыками в различных областях: от создания удобной пользовательской части (интерфейса), до разработки серверной части, взаимодействующей с базой данных и т.д.

Не пугайтесь непонятных слов, ключевое: вы станете специалистом, который способен с нуля самостоятельно разработать веб-приложение и развернуть его на сервере в сети Интернет.



FullStack

Почему важно уметь самому делать полноценные веб-приложения?

1. Обладая такими навыками, вы будете понимать, как "всё устроено" и уметь изменять приложение самостоятельно (а не ждать, пока найдётся нужный специалист)
2. В современном мире очень важна скорость – вы можете придумать гениальную идею, но если не сможете её быстро реализовать, то она так и останется идеей – её вполне может реализовать кто-то другой

Поэтому в мы будем рассматривать только то, что имеет практическое значение и позволит вам научиться создавать реальные веб-приложения.



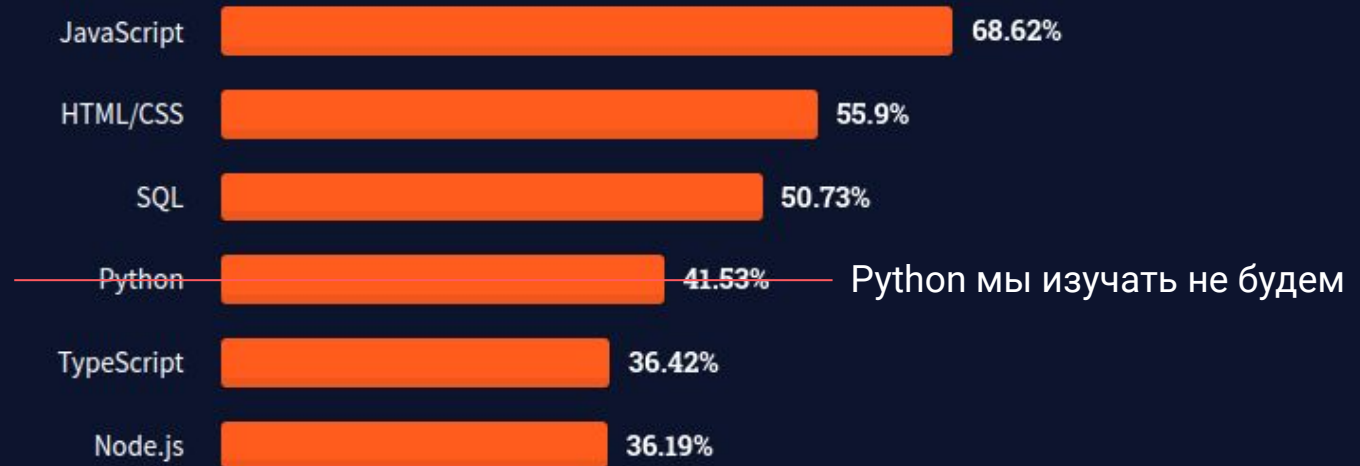
Web

Для этого нам потребуется изучить не только JavaScript, но и сопутствующие технологии:

All Respondents

Professional Developers

58,031 responses



[Исследование StackOverflow](#) – самые популярные технологии 2021.



Web

В рамках данного практикума мы научимся использовать HTML и CSS (заодно разберём, что это такое, как и когда используется, как работает и т.д.), чтобы затем уже приступить к JavaScript (далее – JS), SQL и Node.js.



Инструменты



Инструменты

Для прохождения основной части курса нам понадобятся один инструмент: редактор [VS Code](#) (это специальная программа, в которой мы будем писать код).

Кроме того, важны следующие три момента:

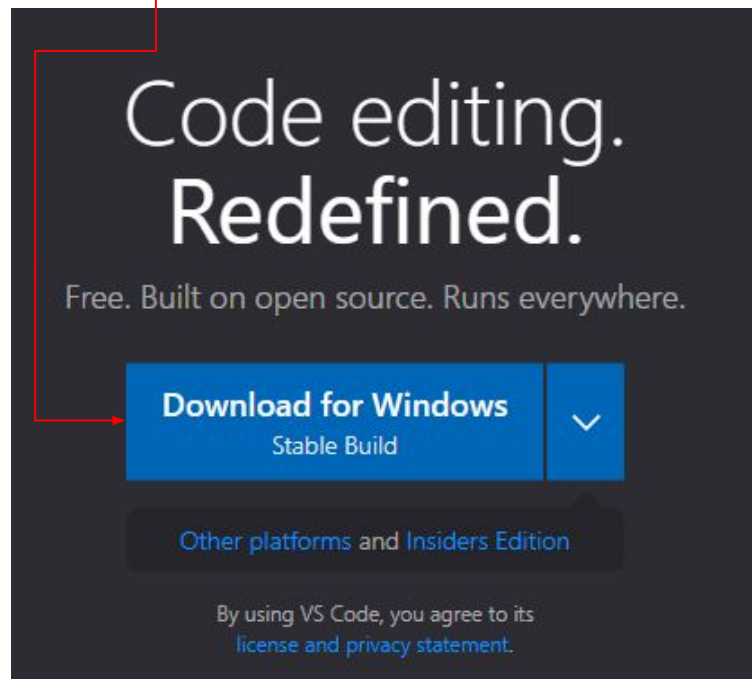
1. У вас должны быть права администратора на компьютере (чтобы вы могли устанавливать программы)
2. Ваш пользователь должен называться по-английски, без пробелов в имени (если это не так – переименуйте)
3. Создавайте все проекты где-нибудь на диске [C:](#), например, в каталоге [projects](#) (следите за тем, чтобы в именах каталогов и файлов не было пробелов, не английских символов и т.д.)

В любом случае, если у вас возникнут проблемы, [пишите в канал](#), мы вам поможем.



VS Code

VS Code самый популярный редактор кода (специальная программа, которая позволяет вам писать код). Перейдите по адресу code.visualstudio.com и скачайте установочный файл для вашей операционной системы. Например, для Windows, нужно выбрать [Download For Windows](#) (если не Windows, то выберите свою ОС из выпадающего списка):



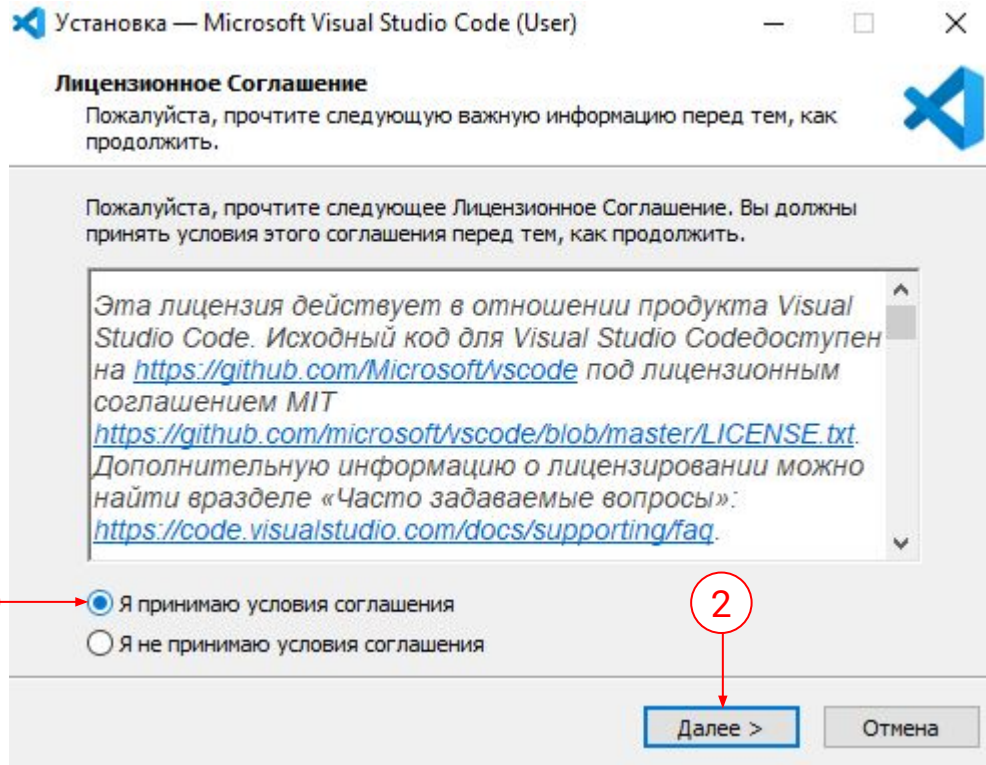
VS Code

Если с официального сайта скачать не получилось, вы можете скачать файл по адресу: <https://alif-skills.pro/media/vscode.exe>.



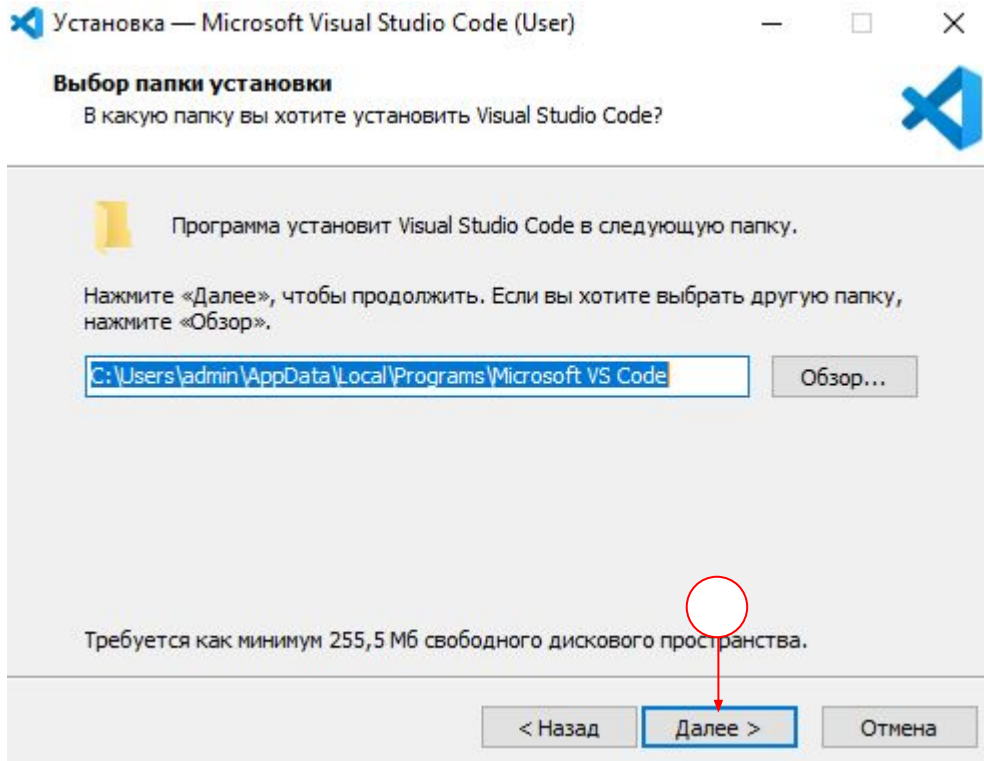
VS Code

Прочитайте и примите условия лицензионного соглашения, после чего нажмите на кнопку **Далее**:



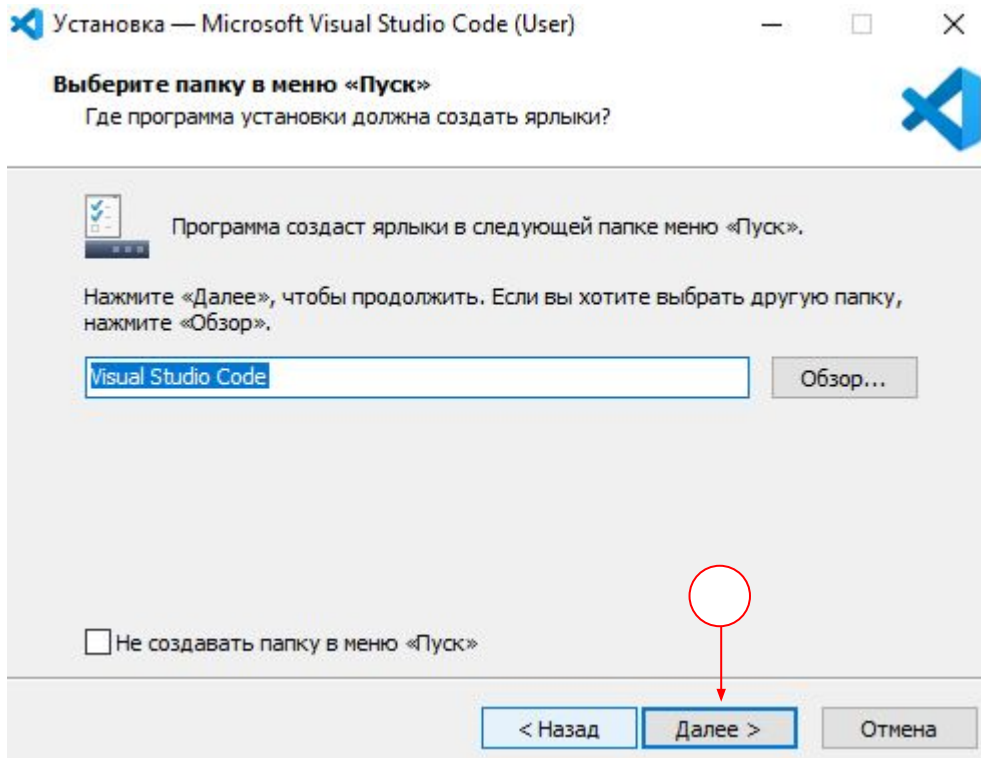
VS Code

Оставьте значения по умолчанию, после чего нажмите на кнопку **Далее**:



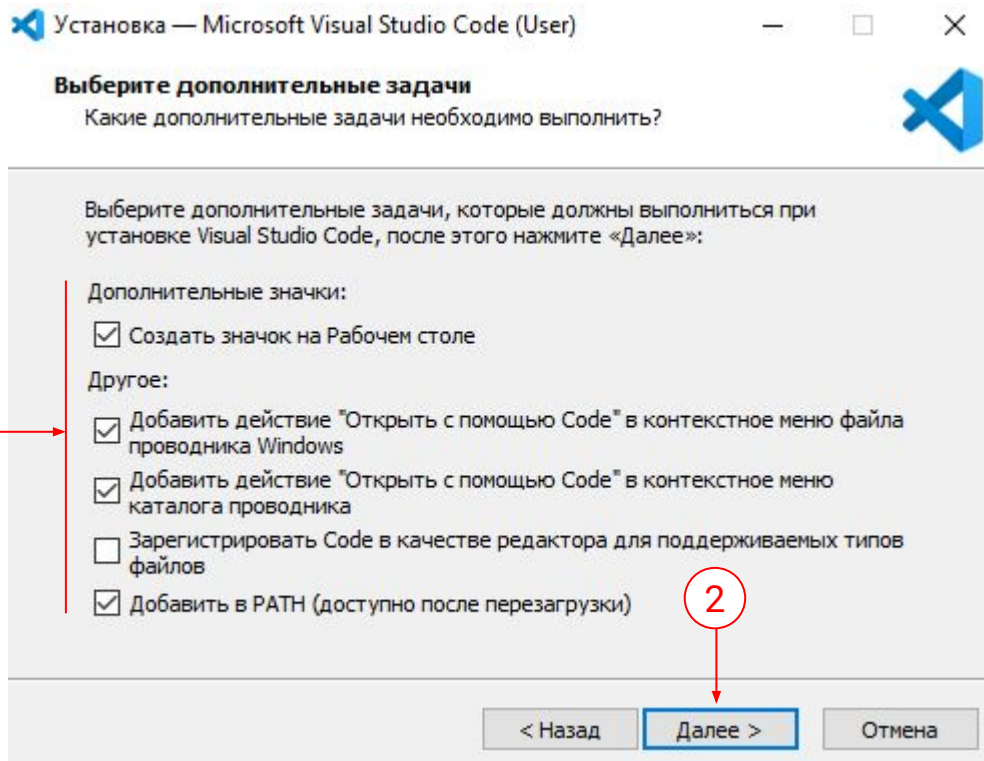
VS Code

Оставьте значения по умолчанию, после чего нажмите на кнопку **Далее**:



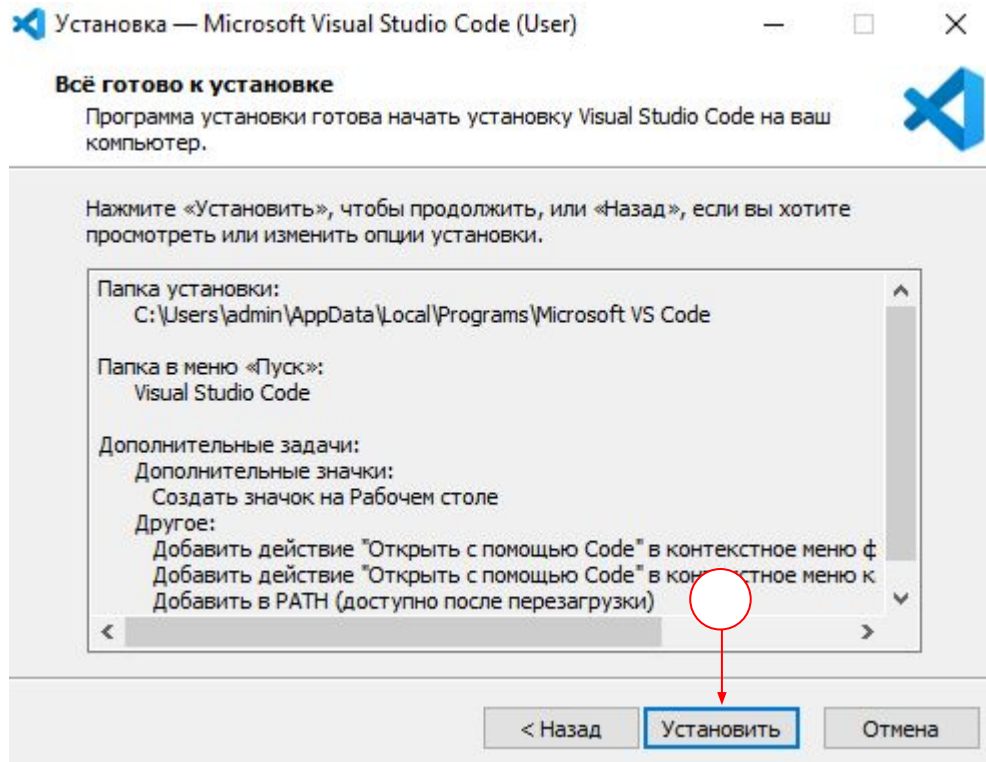
VS Code

Поставьте флажки как на скриншоте, после чего нажмите на кнопку **Далее**:



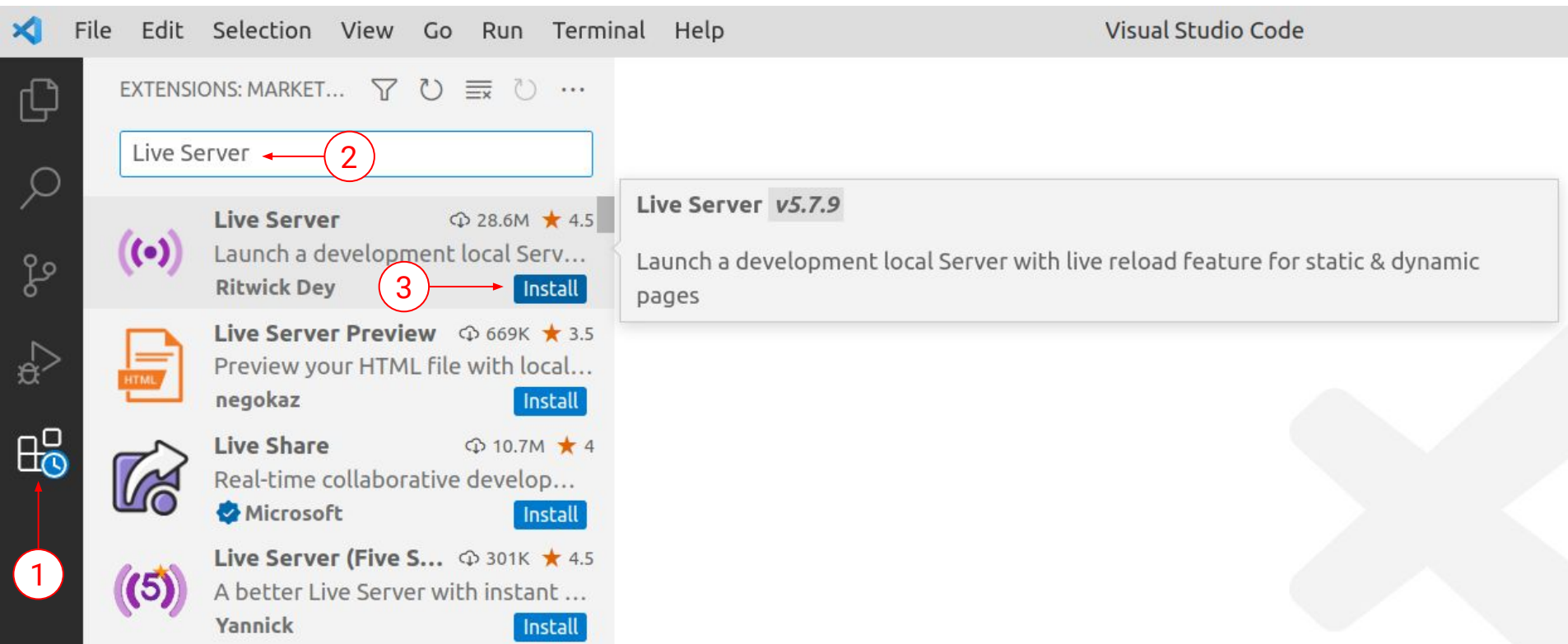
VS Code

Подтвердите установку, нажав на кнопку **Установить**:



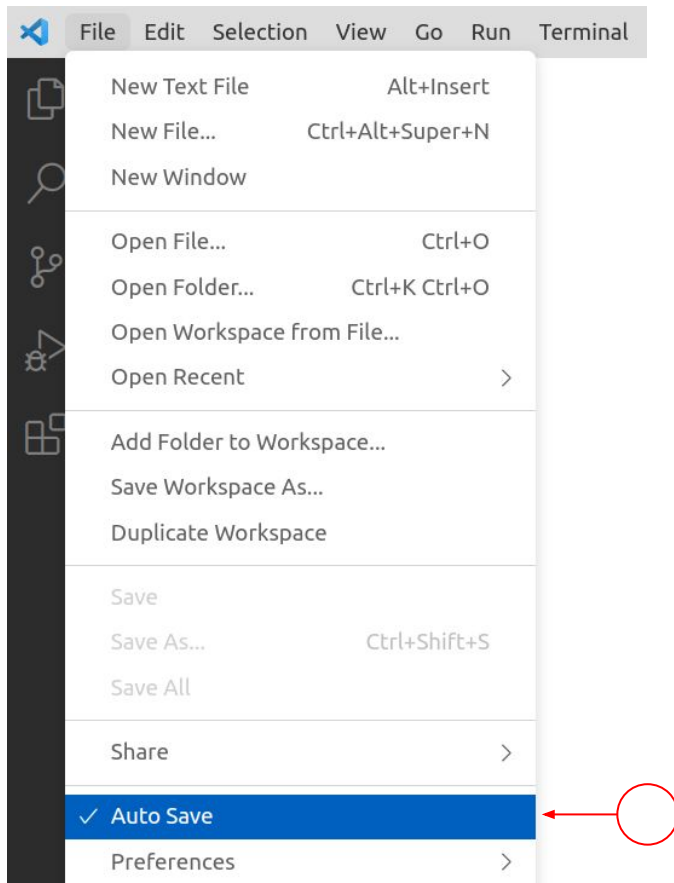
VS Code Live Server

После завершения установки запустите VS Code, зайдите в панельку [Extensions](#) (1),
наберите [Live Server](#) (2) и нажмите на кнопку [Install](#) (3):



VS Code Auto Save

Чтобы изменения, которые вы вносите в файлы автоматически сохранялись, зайдите в пункт меню **File** и выберите **Auto Save**:



Web



Web

Веб-приложение – это клиент-серверное приложение, в которой клиент – это приложение, работающее в веб-браузере.

Давайте разбираться по порядку. Что такое клиент-серверное? Это значит, что само приложение разделено на две части: есть клиент и есть сервер.



Web

В веб-приложениях, клиент – это приложение, работающее в браузере (а браузер работает на вашем компьютере или смартфоне), а сервер (чаще всего) – это приложение, которое расположено на каком-то компьютере, подключенном к сети Интернет.

Эти две части приложения взаимодействуют друг с другом, обмениваясь информацией.



Mobile Application

Нам будет проще понять, если мы будем воспринимать это так же, как мобильные приложения: например, на нашем смартфоне установлен Telegram – для этого мы его скачали и установили. Но чтобы полноценно им пользоваться, нам нужен ещё Интернет – без Интернета мы не сможем отправлять и получать сообщения:

Приложение внутри смартфона



Web Application

С веб-приложениями всё почти так же, но чуточку сложнее: на нашем компьютере или смартфоне установлен браузер, а уже внутри браузера работает приложение:

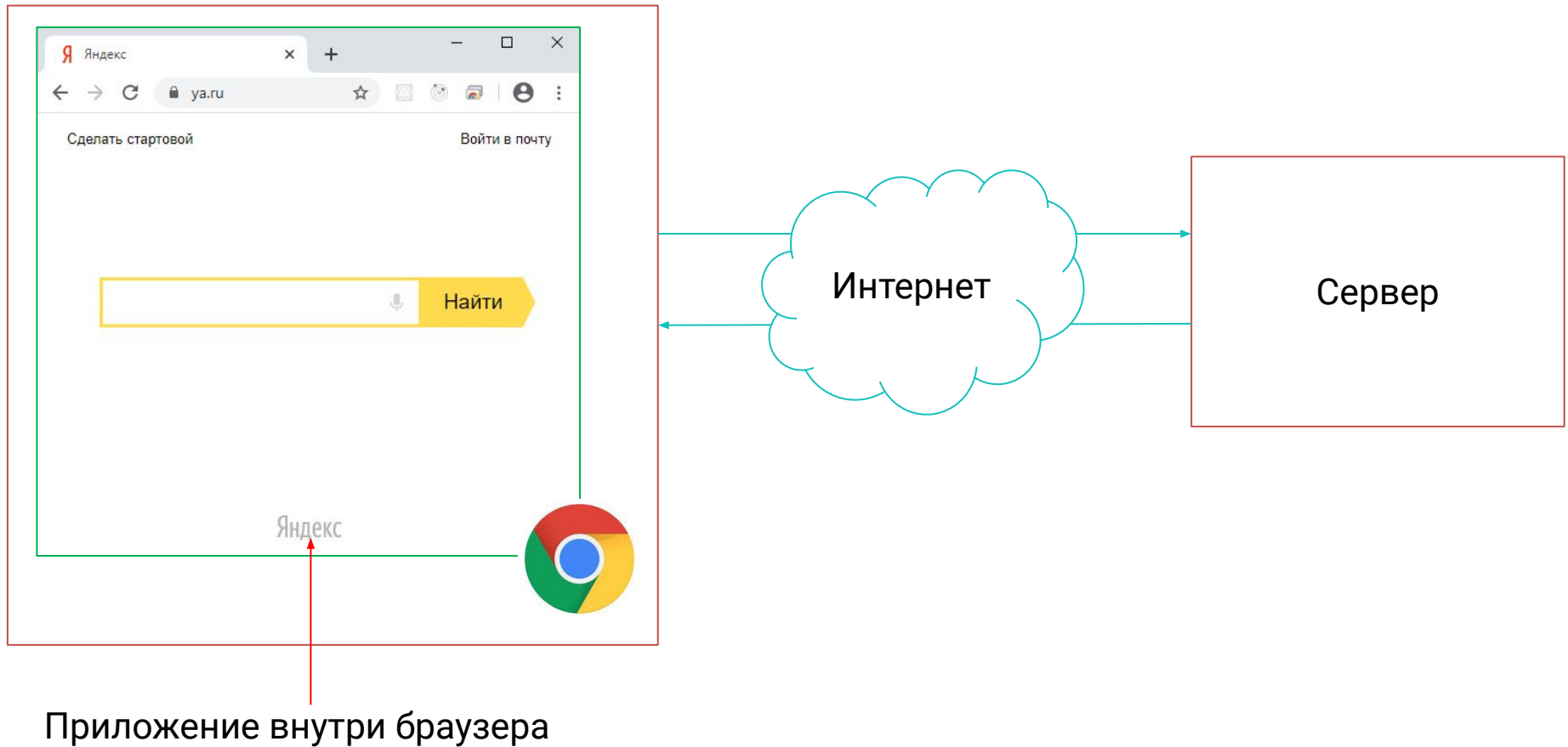


Например, если мы открываем страницу www.ya.ru, то браузер загружает и запускает клиентскую часть приложения (в мобильных приложениях мы делаем это руками* через Google Play или AppStore).

Примечание*: не всегда это необходимо, существует Instant App и аналоги.



Web Application



Упрощения

Нас пока не интересует, что из себя представляет сервер, как браузер загружает и запускает наше приложение и т.д.

Ключевое: мы должны понять самую аналогию – браузер внутри себя запускает программу.

Итого у нас две части одного большого приложения:

1. Внутри браузера (клиент)
2. На сервере (сервер)

Привыкните к тому, что вы не сможете сразу понять как всё устроено: сервер, клиент, передача данных и т.д. Вместо этого учитесь улавливать ключевые идеи и применять их сразу на практике.



Определения

1. **Web (World Wide Web)*** – система связанных веб-страниц, доступных через Интернет. Что значит "связанных"? Мы знаем, что можно переходить от одной странице к другой, используя ссылки (или гиперссылки). Это и есть связность страниц
2. **Веб-страница** – документ, который может быть отображён веб-браузером. Что значит документ? Для простоты будем считать, что документ – это просто файл с расширением [.html](#)
3. **Браузер (веб-браузер)** – программа для просмотра веб-страниц и перехода между ними посредством гиперссылок. Именно браузер, как мы уже с вами говорили, отвечает за то, чтобы запускать клиентскую часть приложения

Примечание*: определения, по большей части, взяты с сайта

<https://developer.mozilla.org>



HTML, CSS, JS

Теперь нам нужно разобраться, как именно делать то, что может запускать браузер. Для этого у нас есть набор технологий:

- **HTML** – позволяет создавать документы, которые умеет обрабатывать браузер* (те самые веб-страницы)
- **CSS** – позволяет визуально оформлять документы
- **JS** – позволяет писать программы, взаимодействующие с документом (и не только)

Давайте разбираться с каждой из них.

Примечание*: строго говоря, браузер умеет обрабатывать огромное количество различных типов файлов (картинки, видео, pdf), но под термином документ (если не указано другое) мы будем понимать именно HTML.



HTML

HyperText Markup Language (HTML) – язык разметки, определяющий структуру документа.

Очень сложное и непонятное определение. Что оно на самом деле значит? "Язык" означает, что существуют определённые правила, которым нужно следовать, чтобы вас поняли. Т.е. мы соблюдаем правила – браузер делает то, что нам нужно, не соблюдаем – никто за результат не отвечает (что нас, естественно, не устраивает).



HTML

Ключевые моменты:

1. Если есть правила, они должны быть где-то описаны. В случае HTML – это [спецификация](#) (или стандарт)
2. Если мы получаем не тот результат, который ожидаем, то виноваты только мы, а не браузер*

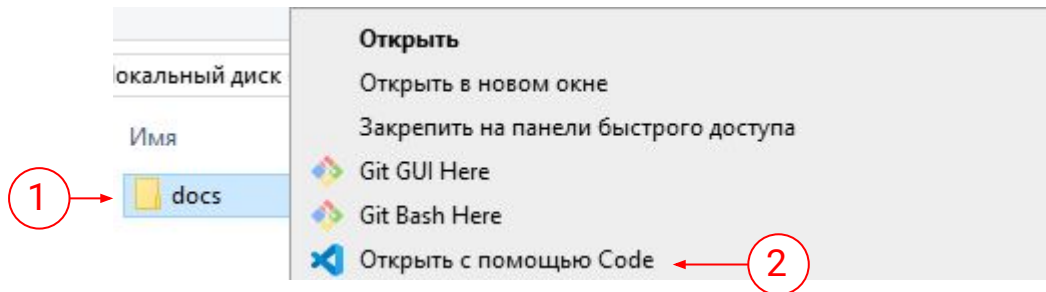
Примечание*: конечно, иногда в браузерах есть ошибки, но чаще будем виноваты именно мы, а не ошибка в браузере.



HTML

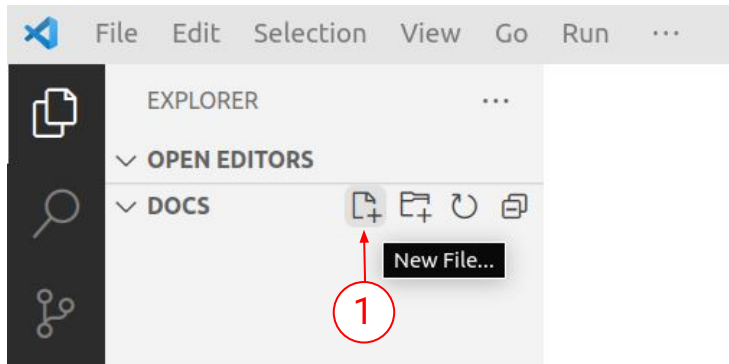
Мы с вами будем детально разбираться HTML и CSS в рамках этого курса.

Создадим в каталоге наших проектов каталог **docs** и откроем его в VS Code (клик правой кнопкой мыши на каталоге):

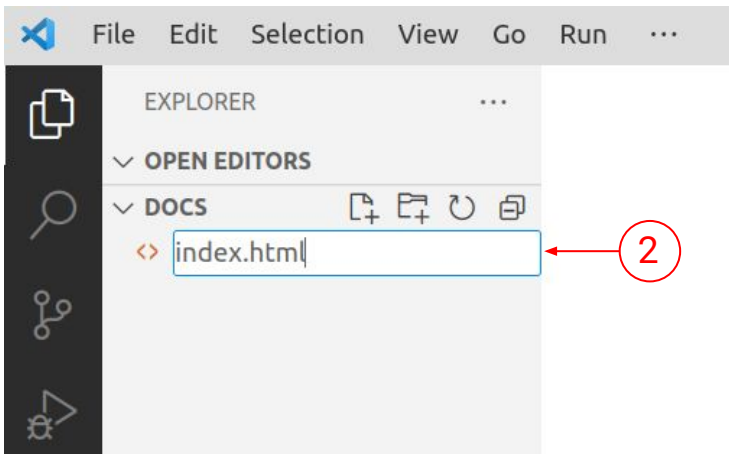


HTML

В боковой панельке нажмите на **+** для создания нового файла:

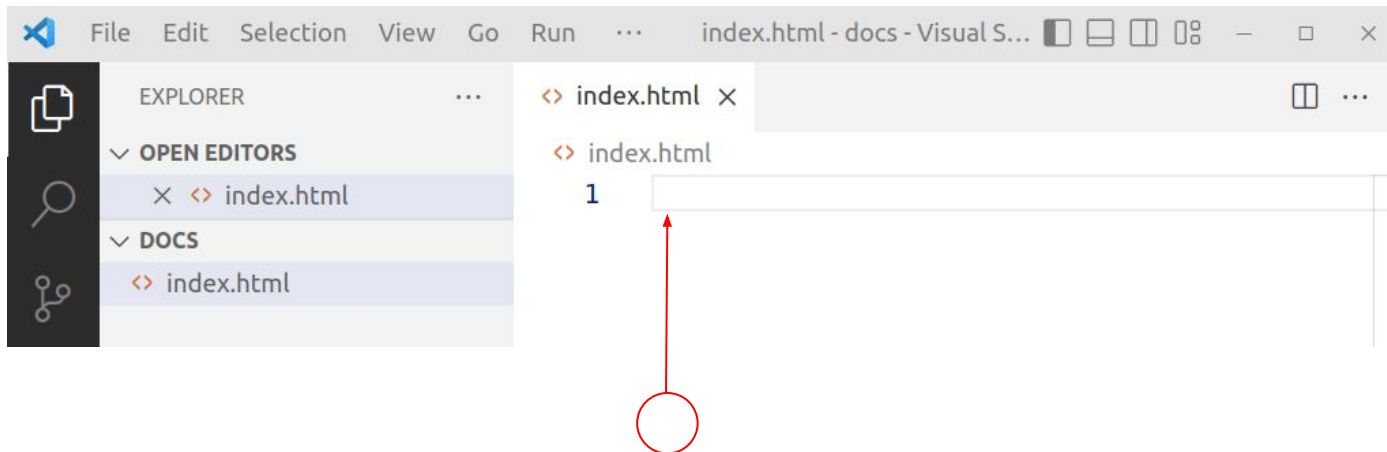


Введите **index.html** и нажмите на **Enter**:



HTML

После этого файл откроется для редактирования в основной части редактора:



Там мы и будем писать код.



index.html

`index.html` – общепринятое имя для основного файла вашего веб-приложения.

Несмотря на то, что можно использовать и другое имя, считается хорошим правилом следовать идее "Convention over Configuration" (следуем общепринятым соглашениям вместо того, чтобы что-то настраивать).



HTML Document



HTML Document

Базовая структура HTML-документа выглядит следующим образом (пока этот код набирать не нужно):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello, Alif Skills!</h1>
</body>
</html>
```

Давайте разберёмся с основными правилами.



HTML Document

Весь документ состоит из HTML элементов и некоторых других сущностей. У нас из таких "специальных" сущностей есть только `<!DOCTYPE html>` (произносится как "доктайп"), с которой и начинается наш документ.

`<!DOCTYPE html>` – это специальная конструкция, которая сообщает браузеру, что этот документ стоит обрабатывать как HTML самой последней версии.

Весь остальной документ состоит из набора HTML элементов.



HTML Element

HTML элемент представляет из себя специальную запись:



Пишется всё именно в таком порядке: сначала открывающий (`<h1>`), затем содержимое (в данном случае – текст `"Hello, Alif Skills!"`) и только потом закрывающий (`</h1>`).

Закрывающий тег от открывающего отличается наличием символа `/`.



HTML Element

Все допустимые элементы (те, которые понимает браузер) описаны в [спецификации](#). Элементы – это те "кирпичики", из которых строится страница.

Например, есть элементы для отображения картинок, аудио, видео. Элемент **h1** предназначен для отображения заголовка первого уровня (самого главного заголовка на странице). Помимо него есть ещё **h2**, **h3**, **h4**, **h5**, **h6** (представьте, что мы пишем реферат, курсовую или диплом – у нас есть название всего документа, название раздела, подраздела и т.д., в HTML документе всё так же).

Важно: мы пока не разбираем, какие есть элементы и для чего они нужны (всё будем делать по-порядку). Сейчас наша задача – понять общие правила того, как всё устроено.



Вложенность

Элементы можно вкладывать друг в друга, т.е. внутрь элементов можно помещать текст и другие элементы:

```
<body>  
| <h1>Hello, Alif Skills!</h1>  
</body>
```

Здесь внутри элемента `body` расположен элемент `h1`. Это позволяет организовывать дерево элементов: тот элемент, в который вкладывают другие, называется **родительским элементом** (`body` родительский элемент для `h1`), а те, которые вкладывают, называют **дочерними элементами** (`h1` дочерний элемент для `body`).

Важно правило: у каждого элемента есть только один родительский элемент.



Вложенность

Очень важно соблюдать правило вложенности – элемент не может "выходить" за границы родительского элемента.

Пример правильной вложенности:

```
<body>  
| <h1>Hello, <span>Alif Skills</span>!</h1>  
</body>
```

Элемент не выходит за "границы" родительского

Примеры неправильной вложенности (внимательно следите за вложенностью!):

```
<body>  
| <h1>Hello, <span>Alif Skills!</h1></span>  
</body>
```

Элемент **выходит** за "границы" родительского

```
<body>  
| <h1>Hello, <span>Alif Skills!</h1>  
</body>
```

Отсутствует закрывающий тег



Q & A

Q: что делает браузер, когда встречает неправильно оформленный документ (например, не соблюдено правило вложенности)?

A: он исправляет документ так, как считает нужным. **НО:** он может исправить его не так, как нужно нам! Поэтому важно сразу писать правильно оформленные документы.



Закрывающий тег

Для некоторых элементов работает следующее правило: если у элемента нет содержимого (текста внутри или вложенных элементов), то можно использовать сокращённую запись:

```
<meta charset="UTF-8"/>
```

Либо вообще опускать:

```
<meta charset="UTF-8">
```

Мы разберём чуть позже для каких элементов это правило работает.



Атрибуты

Элемент может содержать атрибуты (располагаются только в открывающем теге):

`<meta charset="UTF-8">`

имя
атрибута

значение

Атрибуты позволяют "настраивать" свойства элемента. Например, через атрибуты можно будет задать "громкость" аудио или видео-элемента, ширину, высоту и т.д.



Emmet

Мы с вами разобрали ключевые моменты приведённого фрагмента HTML-кода.

Самый главный вопрос: "Как эту базовую структуру создать"? Ведь запомнить её с ходу достаточно сложно.



Emmet

На самом деле, запоминать её и не нужно, потому что вам будет помогать специальный инструмент [Emmet](#) (который уже встроен в VS Code). Наберите в редакторе символ **!** и подождите, пока появится подсказка, после чего нажмите клавишу **Tab**:



Если вдруг подсказка пропала или не появляется, нажмите на клавиатуре **Ctrl + пробел** (плюс набирать не надо, это значит, что нужно одновременно нажать **Ctrl** и **пробел**). Скрыть подсказку можно с помощью клавиши **Esc**.



Emmet

Emmet за нас напишет типовую структуру:

<> index.html > html > head > meta

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Document</title>
8  </head>
9  <body>
10  |
11 </body>
12 </html>
```

у вас может не быть этой строки (это не критично)

Нажимая на клавишу **Tab** мы можем перемещаться между подсвеченными областями, пока не попадёте внутрь элемента **body**.



Emmet: элементы

Помимо базовой структуры, мы можем с помощью Emmet создавать и элементы.

Для этого нужно написать название тега элемента и нажать на **Tab**:

```
<body>  
  h1  
</bo  h1 Emmet Abbreviation  
</html>
```

Нажатие на **Tab** создаст открывающий и закрывающий тег + поместит курсор между ними

```
<body>  
  <h1>|</h1>  
</body>  
</html>
```

Теперь внутри элемента мы можем написать **Hello, Alif Skills!**



Emmet

Emmet имеет ещё кучу других возможностей, с которыми мы познакомимся в рамках курса.

Например, он знает, для каких элементов закрывающий тег не обязателен и не будет его писать (а вам не придётся это запоминать, если вы пользуетесь Emmet). Чтобы быстро освоить Emmet (как и любой другой инструмент), принято пользоваться [шпаргалками](https://alif-skills.pro/media/emmet.pdf) (cheatsheets). Если документ по ссылке не доступен, скачайте его с <https://alif-skills.pro/media/emmet.pdf>.

Важно: привыкайте с первых дней меньше писать и больше пользоваться инструментами. Так вы будете гораздо продуктивнее.



Работа браузера



Live Server

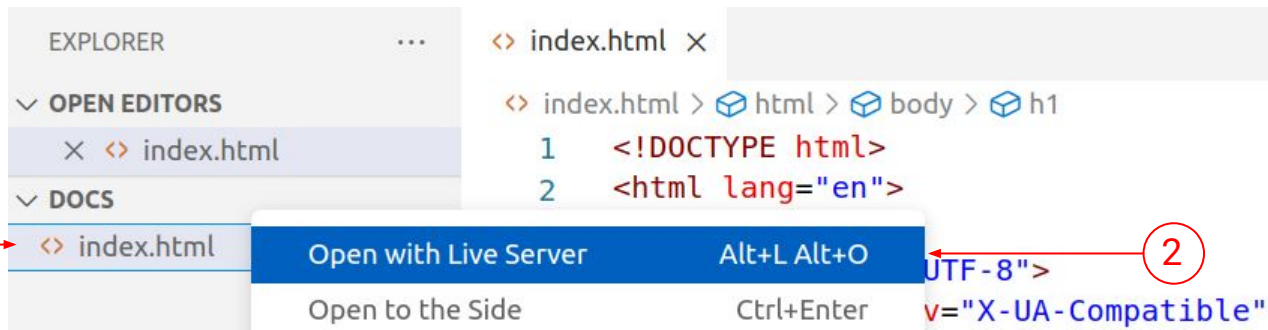
Мы с вами создали документ, теперь нам нужно научиться "загружать" его в браузер и смотреть на результат.

Можно, конечно, открыть его просто так (пойти в каталог с файлом и открыть двойным кликом), но это неправильный подход. Напоминаем, мы хотим добиться следующей схемы:



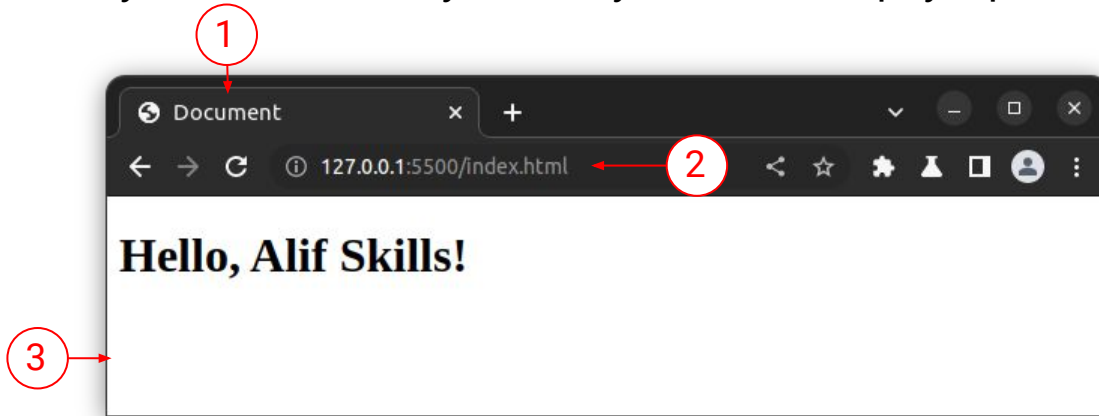
Live Server

Щёлкните правой кнопкой мыши на `index.html` (1) и выберите `Open with Live Server` (2):



Live Server

В установленном у вас по умолчанию браузере откроется следующая страница:



Ключевые вопросы:

1. Откуда взялось название вкладки (1)?
2. Что за адрес в адресной строке (2)?
3. Почему из всего кода, что мы написали, мы видим только содержимое **h1** (3)?

Важно: если у вас пустая белая страница, убедитесь, что вы написали внутри **h1** текст и сохранили страницу (включили авто-сохранение).



HTML

В HTML документе есть две большие "части": **head** и **body**.

head отвечает за мета-данные страницы (т.е. специальные служебные данные для браузера) – например, название вкладки (**title**) или масштаб.

body же отвечает за те данные, которые будут отображены на самой странице, именно поэтому мы видим содержимое **h1**.

<> index.html > ...

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta http-equiv="X-UA-Compatible" content="IE=edge">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <title>Document</title>
8  </head>
9  <body>
10   <h1>Hello, Alif Skills!</h1>
11 </body>
12 </html>
```

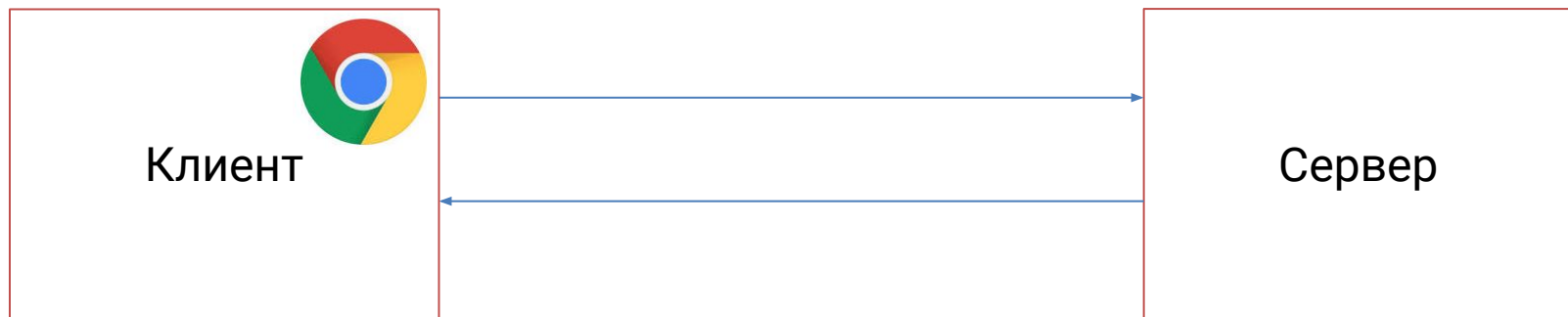


Live Server

При установке VS Code, мы установили расширение Live Server, которое создаёт сервер прямо на нашем компьютере. И этот сервер работает по адресу:

<http://127.0.0.1:5500>.

При этом нам даже не нужен Интернет и общая схема нашей работы будет выглядеть вот так (всё работает внутри одного компьютера):



Live Server полезен ещё тем, что автоматически обновит страницу в браузере, если мы туда внесём изменения (попробуйте), т.е. мы сразу будем видеть все изменения.



Live Server

У вас может возникнуть вопрос "Зачем нам Live Server, если мы и без него можем открыть страницу в браузере и обновлять?".

Причин несколько:

1. Обновлять руками – неудобно
2. Браузер запретит вам использовать некоторые возможности из JS в дальнейшем, если вы будете открывать HTML-документ просто двойным кликом в файловом менеджере

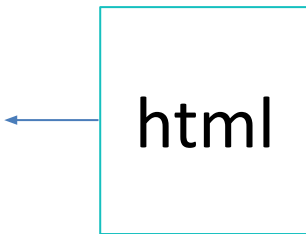
Поэтому обязательно используйте Live Server. Все лекции будут построены на том, что вы смотрите на результат работы именно так, а не открывая файл двойным кликом.



Как работает браузер

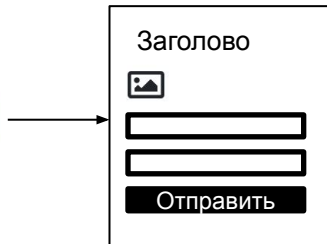
Теперь давайте разберёмся с тем, как работает браузер: браузер скачивает по адресу документ и обрабатывает его в соответствии со спецификацией, превращая документ в визуальное отображение для пользователя:

1



загрузка документа

2



визуальное отображение



Завершаем работу

Для завершения работы достаточно закрыть окно VS Code (Live Server автоматически прекратит свою работу). Браузер придётся закрыть самостоятельно.



ИТОГИ



ИТОГИ

В этой лекции мы обсудили достаточно много важных моментов:

1. Установку инструментов
2. Основы работы в VS Code
3. Отображение страницы с помощью Live Server в браузере

В следующих занятиях мы будем опираться на то, что вы уже изучили в это занятие (и не будем детально описывать процессы создания файлов, запуска Live Server и т.д.).



ДОМАШНЕЕ ЗАДАНИЕ



Орг.моменты

Практикум состоит из 8 обязательных занятий. Мы выкладываем новые занятия каждую субботу в 12:00 (по Душанбе).

Каждую пятницу в 23:59 (по Душанбе) дедлайн сдачи домашнего задания. Дедлайн – это предельный срок, до которого вы должны сдать ДЗ.

Если не успеете сдать в срок домашнее задание, тогда этот практикум будет для вас закончен и вы сможете зарегистрироваться на запуск следующего через несколько месяцев.

Все вопросы вы сможете задавать в [Телеграм канале](#).



Орг.моменты

Дата публикации материалов и дедлайн пишется в личном кабинете рядом с каждым занятием:

#1 Web

Дедлайн: 16.12.2022 23:59:59

Материалы будут доступны: 10.12.2022 10:00:00



ДЗ: Hello, HTML!

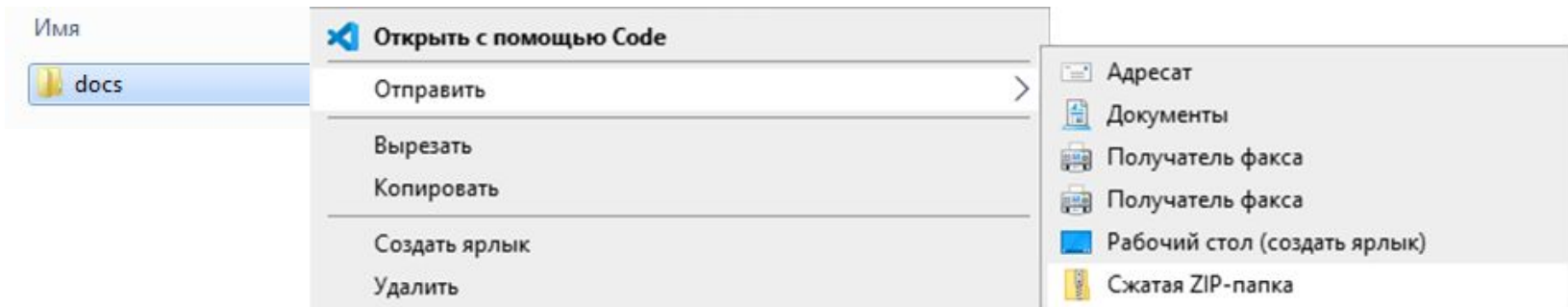
Создайте проект аналогично тому, как мы это делали на лекции. В `h1` разместите текст: `Hello, HTML!`

Проект должен располагаться в каталоге `docs` (именно его и нужно заархивировать).



Как сдавать ДЗ

Вам нужно запаковать в zip-архив ваш каталог с проектом (не содержимое каталога, а сам каталог) – выделяете его и выбираете Отправить → Сжатая ZIP-папка:




Полученный архив загружаете в личном кабинете пользователя.

Важно: учитывается только последняя отправленная попытка.



Как сдавать ДЗ

Перейдите в личный кабинет на сайте <https://alif-skills.pro> и выберите **Загрузить решение:**

#1 Web

Дедлайн: 16.12.2022 23:59:59

[Смотреть лекцию](#) [Загрузить решение](#)



Как сдавать ДЗ

На открывшейся странице выберите нужную задачу и нажмите [Загрузить решение](#) (вам предложат выбрать ваш zip-файл):

Решения

Hello, HTML!

Баллов: 1 Обязательна: Да Дедлайн: 16.12.2022 23:59:59

 [Загрузить решение](#)

Только файлы формата zip, размером не более 1 мегабайта.

Важно: учитывается **только последняя попытка**.

Нет попыток

[Назад](#)



Неверное решение

Если вы загрузите неверное решение, то в поле статус будет написано **НЕ ПРИНЯТО** (1) и вам нужно будет загрузить исправленное решение (2):

2 →

Загрузить решение

Только файлы формата zip, размером не более 1 мегабайта.

Важно: учитывается **только последняя попытка**.

✓ Последняя попытка

1 ↓

#7 Статус: **НЕ ПРИНЯТО** Отправлено: 10.12.2022 00:58:47 Проверено: 10.12.2022 00:58:56

```
START
Распаковываем архив - OK
Проверяем каталог docs - FAIL
В вашем архиве не найден каталог docs
docs directory not found

FAIL
Проверки завершились с ошибкой.
```

Количество попыток не ограничено.



Как читать ошибки

Открываете результаты и идёте сверху вниз (1) до первой записи **FAIL** (2):

Загрузить решение

Только файлы формата zip, размером не более 1 мегабайта.

Важно: учитывается **только последняя попытка**.

✓ Последняя попытка

```
#7 Статус: НЕ ПРИНЯТО Отправлено: 10.12.2022 00:58:47 Проверено: 10.12.2022 00:58:56

1 START
  Распаковываем архив - OK
  Проверяем каталог docs - FAIL ← 2
  В вашем архиве не найден каталог docs
  docs directory not found

  FAIL
  Проверки завершились с ошибкой.
```

Ниже будет написана причина ошибка – "В вашем архиве не найден каталог docs", т.е. мы забыли в архив положить папку docs или назвали её неправильно.



Как читать ошибки

Если ошибка будет в структуре вашего HTML, то:

- + как должно быть
- как у вас

```
START
Распаковываем архив - OK
Проверяем каталог docs - OK
Проверяем файл index.html (в каталоге проекта) - OK
Проверяем приложение
Пишем тест - OK
Запускаем тесты - FAIL
FAIL ./test.js
  × Проверка документа (238 ms)

    • Проверка документа

      + Версия бота
      - Ваша версия

        <body>
      +   <h1>Hello, HTML!</h1>
      -   <h1>Hello, Web!</h1>

        </body>
```



Как читать ошибки

В некоторых случаях бот будет от вас требовать определения атрибутов в определённой последовательности – обращайтесь на это внимание!



Успешная сдача

После того, как вы успешно выполните все требования задания, оно будет отмечено как зачтённое (пересдать его будет нельзя):

Hello, HTML!



Баллов: 1 Обязательна: Да Дедлайн: 16.12.2022 23:59:59

Задача сдана

✓ Последняя попытка

#30 Статус: **ПРИНЯТО** Отправлено: 10.12.2022 04:06:47 Проверено: 10.12.2022 04:06:51

```
START
Распаковываем архив - OK
Проверяем каталог docs - OK
Проверяем файл index.html (в каталоге проекта) - OK
Проверяем приложение
Пишем тест - OK
Запускаем тесты - OK
OK
```



Успешная сдача

Только после сдачи всех обязательных задач к текущему занятию вам будет доступны материалы следующего занятия.



Спасибо за внимание

alif skills

2022г.

